ALGORITMOS Y ESTRUCTURAS DE DATOS III - 1^er^ Parcial

Fecha examen: 13-MAY-2017 / Fecha notas: a determinar

	Nº Orden	[redacted]	
Completar:	2	[redacted]	Cant. hojas[1]
	Nota (Nº)	[redacted]	8
No completar:	7.6	A	Docente

1. (a) Sea T un árbol que tiene h hojas. Demostrar que $h \neq 1$. 0.5 p.
   (b) Sea B un bosque no trivial. Demostrar que B tiene al menos dos nodos de grado 0, o al menos dos nodos de grado 1. 1.5 p.

2. (a) Demostrar que ningún grafo tiene exactamente 2 árboles generadores. 1 p.
   (b) Para cada $n \in \mathbb{N}$ exhibir un grafo de n vértices que tenga un único árbol generador. Justificar. 0.5 p.
   (c) Para cada par de valores $n \in \mathbb{N}$ y $k \in \mathbb{N}$ con $n \geq k \geq 3$, exhibir un grafo de n vértices que tenga exactamente k árboles generadores. Justificar. 0.5 p.

3. Sea $v = (v_1, v_2, \dots v_n)$ un vector de números enteros. Diseñar un algoritmo que indique la mínima cantidad de números que hay que eliminar del vector para que cada número que permanezca sea múltiplo del anterior (excepto el primero). Por ejemplo, para los vectores $(-5, 5, 0)$, $(0, 5, -5)$ y $(0, 5, -5, 2, 15, 15)$, los resultados deben ser respectivamente 0, 1 y 2. El algoritmo debe tener complejidad temporal $O(n^2)$ y estar basado en programación dinámica. Mostrar que el algoritmo propuesto es correcto y determinar su complejidad (temporal y espacial). Justificar. 2 p.

4. Brian está tan especializado en su trabajo que gana fortunas con cada cosa que hace. Como contrapartida, sólo algunas empresas requieren sus servicios. Brian conoce las empresas que podrían convocarlo, así como las ciudades donde están ubicadas. Un conjunto de rutas comunican las ciudades entre sí, y cada ruta cuenta con una estación de peaje. Usualmente cada estación de peaje cobra cierto importe a cualquiera que transite por la ruta correspondiente, pero algunas rutas están tan subsidiadas que entregan dinero a los viajeros en vez de cobrarles. Brian no cree en la redistribución de la riqueza, por lo que quiere vivir en una ciudad tal que para cada empresa exista al menos una manera de ir a la misma con el importe neto de los peajes a su favor. 2 p.

   Hay n ciudades, p ciudades que tienen empresas, y m rutas. Cada ciudad se identifica por un entero distinto entre 1 y n. Cada ruta conecta determinado par de ciudades y se puede recorrer desde la primera hacia la segunda. Se puede ir de cualquier ciudad a cualquier otra recorriendo las rutas, pasando eventualmente por ciudades intermedias.

   Diseñar un algoritmo eficiente que decida si existe alguna ciudad en la que Brian quiera vivir. La entrada del algoritmo es la cantidad n de ciudades, la cantidad p de ciudades que tienen empresas, la cantidad m de rutas, la lista de ciudades que tienen empresas, y para cada ruta su ciudad inicial, su ciudad final y el importe de su peaje (negativo cuando la estación de peaje entrega dinero en vez de cobrarlo). Mostrar que el algoritmo propuesto es correcto y determinar su complejidad. Justificar. El mejor algoritmo que conocemos tiene complejidad $O(n^3)$.

5. (a) Sea G_n un grafo *conexo* de n vértices que tiene un único par de vértices de igual grado (notar que $n \geq 2$). Demostrar que G_n tiene al menos un vértice *universal*, y que si $n \geq 3$ entonces tal vértice es único. 0.5 p.
   SUGERENCIA: Para la existencia, por absurdo concluir que hay $n-2$ valores posibles para los n grados. Para la unicidad, ídem.
   (b) Sea H_n un grafo *no conexo* de n vértices que tiene un único par de vértices de igual grado. Demostrar que H_n tiene al menos un vértice *aislado*, y que si $n \geq 3$ entonces tal vértice es único. 0.5 p.
   (c) Demostrar que salvo isomorfismos, existe a lo sumo un grafo conexo de n vértices que tiene un único par de vértices de igual grado, y a lo sumo un grafo no conexo de n vértices que tiene un único par de vértices de igual grado. 1 p.
   SUGERENCIA: Inducción con absurdo en el paso inductivo.

---

[1] Incluyendo a esta hoja. Entregar esta hoja junto al examen.

Nº: 2

2 p

Ejercicio 1:

a) • Si T es el árbol trivial empíricamente podemos ver que la cantidad de hojas es $\neq 1$ ya que no tiene hojas y un solo nodo de grado 0

• Si T no es el árbol trivial sabemos por la práctica que $\forall$ T' árbol tq T' es dif del árbol trivial T' tiene al menos 2 hojas, lo que generan un absurdo si asumimos que $\exists$ algún árbol dif del trivial tiene exactamente una hoja.

b) Sea B un bosque, podemos separarlo en casos

• Si B tiene solo una componente conexa C_1 entonces el árbol T que representa C_1 no puede ser el árbol trivial ya que esto haría a B un bosque trivial y habíamos partido de lo opuesto.

C_1 es conexa y no tiene ciclos por esto en un bosque $\Rightarrow$ es un árbol

Entonces T decimos que es $\neq K_1$ por lo que nuevamente gracias a la práctica podemos decir que tiene al menos 2 hojas, osea 2 nodos de grado 1.

Y como T es un subgrafo de B entonces vimos que en este caso B cumple con tener 2 nodos de grado 1

- Si B tiene ~~dos~~ almenos 2 componentes conexas. podemos tomar a ~~T_1 y T_2~~ $T_1 = C_i$ y $T_2 = C_j$ $i \neq j$ como las dos árboles que representan a 2 componentes conexas de B cualquieras para ver que se cumple la propiedad.

- Si T_1 y T_2 son árboles triviales, entonces listo pasa que B tiene 2 nodos de grado igual a 0 en ~~a~~ $V(B)$. (los nodos que conforman a T_1 y T_2)

- Si ahora T_1 y T_2 (el que no sea árbol trivial) pueden ser ambos no son ambos árboles triviales pasa que alguno de los 2 nuevamente va a cumplir con la propiedad de que tiene almenos 2 hojas. Lo que implica que B tiene almenos 2 nodos de grado igual a ~~en~~ uno.

¿igual 1 y 0? falta caso

0.9 | 0.4 | 0 N°: 2 1.3 p [redacted]

Ejercicio 2:

a)-

- Sea G un grafo con n nodos y $m < n-1$ aristas entonces tenemos que G no es conexo por lo que no tiene ningún A.G

- Sea G un grafo con $n-1$ nodos y $m = n-1$ aristas entonces tenemos que G tiene exactamente un arbol generador. Porque si tuviese más ~~podemos~~ podríamos tomar a T_1 y T_2 como 2 de estos A.G's y decir como $T_1 \neq T_2$ entonces difieren en al menos ~~una arista~~ ~~...~~ una arista e_d. Ahora entonces $T_1 + e_d$ sigue siendo un subgrafo de G porque T_1 ~~...~~ era un A.G de G y $e_d \in G$ tamb. ~~...~~ y tenemos que $\# E(T_1) = n-1$ por propiedades de Árboles entonces $\# E(T_1 + e_d) = n-1+1 = n$ que es mayor a la cantidad de ejes que tenía G en un primer lugar, lo que genera un absurdo por suponer que tenía más de un AG.

- Ahora si G es un grafo de n nodos y $m > n-1$ aristas que $\exists$ al menos $T_1 \neq T_2 \neq T_3$ AG's de G.
$T_1 \neq T_2$, $T_1 \neq T_3 \wedge T_2 \neq T_3$

Sea entonces T_1 un A.G de G como $m > n-1$ y $\# E(T_1) = n-1$ existe $f \in G$ tq $f \notin T_1$ y por la Práctica sabemos que el grafo $T_1 + f$ tiene exactamente un ciclo C, donde

Por definición de Ciclo, al menos tiene $\#C \geq 3$. Por lo que existen e_1 y e_2 $\in T$ + q están en C.
$e_1 \neq e_2 \neq f$

Ahora entonces por el Corolario 6 visto en la Práctica tenemos que si T es un AG de G y f un eje q $f \in G$, $f \notin T$ entonces $T_2 = T + f - e$, siendo e un eje del ciclo ~~que~~ C que se forma al agregar ~~f~~ f a T entonces T_2 es un AG de G también

Ahora entonces volviendo nos queda que dado T_1 podemos formar a $T_2 = T_1 + f - e_1$ y $T_3 = T_1 + f - e_2$ los que nos dejan a 3 árboles generadores de G diferentes entre sí. ¿Por qué son distintos?

¿Cuáles son los pesos de los ejes?

b) Para cada n ~~tomamos~~ armamos K_n que por definición es conexo, y tomamos a $T = Prim(K_n)$, lo que por correctitud de prim nos devuelve a T ~~es~~ un AG de K_n. ~~# por lo~~ Entonces tenemos que T es un grafo con n vértices y $m = n-1$ aristas. Lo que ya vimos en el punto 1 que es un grafo con sólo 1 árbol generador.

1.80

N°:0:2

Ejercicio 3

a → es el arreglo de n posiciones pasado por parámetro

FunciónDinámica:

$$f(i) = \begin{cases} 1 & \text{si } i == 0 \\ \max[\, f(j) \mid 0 \leq j < i,\ a[i] / a[j] \in \mathbb{N} \,] + 1 & \text{c.c} \end{cases}$$

¿si no existe?

Esta función vemos que devuelve la máxima longitud de una lista ~~de~~ tq. cada número sea multiplo del anterior ~~[illegible]~~ tomando como último elemento el número en la posición i.

Algoritmo Botom Up:

la idea va a ser crear un arreglo b de tamaño ~~[illegible]~~ n tq para el final del algoritmo ∀i i<n b[i]=f(i)

el algoritmo lo que va a hacer es una iteración de n pasos sobre el arreglo ~~a~~ ~~[illegible]~~ y comenzando desde la posición 0 iterando crecientemente, y va completando el arreglo b

Nuestro invariante es que ~~[illegible]~~ en la iteración i ∀j<i b[j] = f(i]. Y la respuesta final del ~~nuestro~~ algoritmo es m - ~~max~~ maximoEn(b, n). ~~[illegible]~~ ~~[illegible] la respuesta correcta~~

Esta es la respuesta correcta ya que ~~porque~~ si asumimos que b[i] = f(i) entonces pasa que si m - el max f(i) no es ~~[illegible]~~ la menor cantidad de números que hay que eliminar para que me quede un arreglo donde cada número es múltiplo del anterior.

existía otra secuencia tq eliminando menos números obtenías ~~un arreglo más grande~~ obtenías lo mismo, osea q̄ la long final del arreglo que queda es de M - k. Siendo k la respuesta. Pero si k ≤ m - maxEn(b,n) => ~~[illegible] maxEn(b,n)~~

=> maxEn(b,n) ≤ m-k → Pero es un absurdo que haya un arreglo más largo ya que maxEn(b,m) →

nos devolvía el más largo que cumplía la condición.

Y la forma que calculamos b[i] es correcta ya que ~~en el algoritmo exactamente nos fijamos el maximo~~

Si asumimos que hasta i-1 era correcta entonces b[i] exactamente se fija el maximo de los b[j] con j<i que cumplen que ~~a[j]~~ a[i] % a[j] == 0 lo que por def es b[i]. Y empezamos desde el caso base que es trivialmente cierto.

```
MaxSecMul (in a: arreglo<int>, int n)

    b = arreglo[n]
    b[0] = 1.
    for(int i=0; i < n; i++) {
        int max=0;
        for (int j=0; j < i, j++) {
            if ( a[i] % a[j] == 0) {
                if (b[j] > max) max = b[j]; // O(1)
            }
        }                    // ciclo de a lo sumo n veces
        b[i] = max+1;        // O(1)
    }                        // ciclo de a lo sumo n iteraciones

    int casiRes = MaximoE. (b, n];
    return n - casiRes;
}
```

Complejidad: Es $O(n^2)$ temporalmente y $O(n)$ en memoria.

En memoria porque creamos un arreglo de O(n) nomas y después una cant constante de variables. Y temporal porque son todas operaciones constantes metidas en 2 ciclos de O(n) iteraciones, por lo que el ciclo interior se va a repetir n veces también.

Nº.0: 2

Ejercicio 4:

El modelado del Problema es obviamente con digrafos, designo a cada ciudad como nodo y a cada ruta como arista, poniendo como peso de la arista el precio del Peage.

El algoritmo se basa de a la matriz de adyacencias con la cual represento el grafo transformandola en la matriz de caminos minimos entre ~~[illegible]~~ cada par de nodos que se devuelve el algoritmo de Floyd. Que lo puedo utilizar justamente porque por aclaraciones del problema ~~[illegible]~~ la representación de cualquier instancia es un digrafo, conexo, sin ciclos negativos.

Bueno y luego de que se tiene dicha Matriz simplemente queda chequear para cada ciudad si alguna cumple con tener caminos gratuitos hacia todas aquellas que tengan una empresa.

El algoritmo es correcto porque gracias a la correctitud del algoritmo de floyd puedo asegurar que los caminos minimos se encuentran bien representados y luego si una ciudad es devuelta como resultado, es como consecuencia de un chequeo trivial sobre esta información.

La Complejidad es de $O(n^3)$ ya que hay tres secciones con complejidad dif de $O(1)$

Cargar la matriz es $O(n^2)$

Algoritmo de Floyd $O(n^3)$

Chequear si Alguna Ciudad Cumple $O(n^2)$ ✓

$$O(n^2) + O(n^3) + O(n^2) = O(n^3)$$

Algoritmo

¿POR QUE SIRVE CHEQUEAR CAMINOS MÍNIMOS? NO ES TRIVIAL

Nº0:2

Ejercicio 4º:

Algoritmo:

n ← cant ciudades
p ← cant Empresas
m ← cant Rutas
Emp ← Arreglo con las ciudades con Empresas (long p)

```
Función () {

int n ; cin >> n;
int p ; cin >> p;          // ciudades con empresas

int m; cin >> m;

int Emp [p];
  for (int i=0; i<p; i++)          // que ciudad tiene empresa.
       cin >> Emp[i];
}

int matriz [n][n] ;         → Poner en ceros
for (int i=0; i<m; i++) {   → Ingreso las rutas

   int a,b,p;               // a = ciudad inicial del eje
   cin>> a; cin>>b; cin>> p;   b = ciudad final
                               p = Precio Peaje
   Matriz [a][b] = p
}
```

TENDRÍA QUE SER $m[i][i]=0$
$m[i][j] = +\infty \ \forall i \neq j$

→ OJO, NO TE QUEDAN EN $+\infty$ LAS ARISTAS QUE NO ESTÁN

Algoritmo Floyd (Matriz, n) // Transforma mi matriz en una Matriz de caminos mínimos entre cada ciudad, se puede ejecutar sin problemas ya que no hay circuitos negativos

↓

```
int res = -1
for(int i=0; i<n; i++){ bool cumple = true
    for (int j=0; j<P; j++){
        if (M[i][j] >0) cumple = false;
    }
    if (cumple) res = i;
}

return res;
```

↳ Emp[j]

// $P < n$

// ⇒ toda esta sección está acotada por $O(n^2)$.

N°0:2 4:6/7

Ejercicio 5

1

a)

- Si G tiene solo 2 nodos, el único grafo que puede ser es v_1 — v_2 que trivialmente cumple la propiedad ya que solo hay un par de nodos de igual grado (v_1, v_2) y existe un nodo universal el mismo.

- Si G tiene 3 o más nodos, tenemos que: Sea D igual al conj de grados posibles de cada nodo. D es igual a $\{1, \dots, n-1\}$ con $\#D = n$ ya que como es conexo no trivial todos los nodos tienen grado mayor o igual a 1. Luego como queremos que solo exista un par de nodos v_{k_1}, v_{k_2} tq tengan el mismo número de grado k, vamos a tener un valor de D reservado.

  - Si $k = n-1$. En este caso pasa que tenemos 2 nodos universales, lo que implica que ahora todo el resto de los nodos tienen grado mayor o igual a 2, porque están conectados con v_{k_1}, v_{k_2}. Lo que deja a $D = \{2, \dots, n-2\} \Rightarrow \#D = n-3$ porque $n \geq 3$

    Ahora si asumimos que es posible esto y que sin embargo solo haya un par de nodos con igual par, significa que cada uno de los ~~nodos~~ $n-2$ nodos restantes van a tener un grado diferente, pero pasa que hay solo $n-3$ valores posibles, lo que hace absurdo ~~el que existe una numeración para cada nodo~~

el que haya un solo par de nodos iguales y 2 nodos universales.

- Ahora para $k \neq n-1$, vamos que no pueden haber 2 nodos universales y que se cumpla la condición de que haya solo un único par de nodos con el mismo grado, por lo que queremos ver ahora es que ~~haya~~ no haya ninguno, o sea el grado $n-1$ no es válido. por lo que nuestro D pasaría a ser $D = \{1, \dots n-1\} - \{k, n-1\}$ es $\# D = n-3$.

Nuevamente llegamos a que si asumiendo ahora en este caso que no puede haber nodos universales, ~~[illegible]~~ y aún así haber solo 2 nodos con igual grado ~~[illegible]~~ nos quedamos con que para que se cumpla esto debería pasar que $n-3$ valores se repartan en $n-2$ nodos sin generar repeticiones, lo que es absurdo.

b) • Δca nuevamente si G tiene 2 nodos ~~...~~ es trivial de comprobar ya que el único grafo con 2 nodos ~~...~~ no conexo es ~~...~~

v_1 v_2

y se comprueba empíricamente que tiene solo 2 nodos de igual grado y al menos un nodo aislado.

• Ahora si G tiene 3 nodos o más podemos decir que si pasa que ~~...~~ se cumple ~~...~~ que solo tiene un par de nodos con igual grado y más de un nodo ~~...~~ aislado, entonces ~~...~~ $\overline{G}$ cumpliría el contraejemplo del punto 5.a) ya que los nodos aislados pasarían a tener grado $n-1$ y ser 2 nodos ~~...~~ universales, y ~~...~~ la ~~...~~ condición de que solo hay solo 2 nodos de igual grado se siguen cumpliendo ya que si $x_1 \neq x_2 \Rightarrow n-x_1 \neq n-x_2$.

Y esto ya probamos que no es posible.

Y si pasa que se cumple que no hay nodos ~~...~~ aislados, ~~...~~ no es conexo, y solo hay 1 par de vertices con igual grado entonces tenemos como en el punto anterior donde ~~...~~ los valores posibles para el grado de un nodo de los $n-2$ nodos diferentes a v_{k1} y v_{k2} son $\{0, \dots, n-2\} - \{k, 0\} \Rightarrow \#D = n-3$

y no hay forma de generar ~~que~~ una repetición de ~~k~~ n-3 valores entre n-2 sujetos sin lograr nuevos repetidos.

c) $P(i)$ = existe a lo sumo un G_i tq G_i es conexo tiene un par de vértices con grado ~~ict~~ igual

y a lo sumo un G_i ~~ea~~ no ~~es~~ conexo tq tiene un par de vértices con grado igual

$CB = P(2)$ = Demostrado en 5a) y 5b).

$P(h-1) \Rightarrow P(h)$:

al menos

~~P(h)⇒~~ · Si G_h es conexo, por la practica sabemos que existen a lo sumo 2 nodos v_1 y $v_2 \in G_h$ tq $G_h - v_1$ es conexo y $G_h - v_2$ es conexo. Entonces tomo a $G' = G_h - v_1$

Entonces tenemos que ~~...~~ G' a lo sumo tiene un par de nodos de igual grado

×

No lo ~~pude~~ supe terminar

Convenía usar los items a) y b): Para el conexo, sacar dos grafos distintos y sacar un vértice <u>universal</u>. Para el no conexo, un <u>aislado</u>